O Misterioso Caso do Git Add Silencioso no macOS: Como a Experiência Superou a Inteligência Artificial

Investigação técnica detalhada sobre como o stat cache do Git e o sistema de arquivos case-insensitive do macOS provocaram um bug silencioso de staging, e como uma solução prática superou o diagnóstico da IA.

Autor: Maicon Santos Webmaicon
Advisor: Lunara LU-WANING 260710

Criado em: 09/06/2026
Atualizado em: 13/06/2026

#Git #MacOS #SoftwareDevelopment #Debugging #CaseSensitivity #StatCache #APFS #GhostEntries #Workaround #ArtificialIntelligence

---

O Misterioso Caso do Git Add Silencioso no macOS: Como a Experiência Superou a Inteligência Artificial

Estava trabalhando em um projeto Laravel (na branch `dg-teste`), modificando quatro arquivos — pequenos ajustes de alinhamento em factories e a renomeação de dois campos. Uma tarefa absolutamente de rotina. Rodei `git add .`, seguido de um `git status`, e... nada. As alterações continuavam aparecendo como modificadas e não preparadas para commit. O `git add` retornava exit code `0` (sucesso absoluto, sem avisos, sem erros), mas se recusava a fazer o staging.

Decidi pedir ajuda para uma IA. Ela veio armada com toda a documentação do mundo e uma confiança inabalável. O que se seguiu foi uma verdadeira autópsia técnica do Git.

O Sintoma: Exit Code 0 com Efeito Zero

```
$ git status
Changes not staged for commit:
  modified: code/back/database/factories/DGPaymentWebhookEventFactory.php
  modified: code/back/database/factories/DGPlanFactory.php
  modified: code/back/database/factories/DGProfileBoostFactory.php
  modified: code/back/database/factories/DGTransactionFactory.php

$ git add code/back/database/factories/DGTransactionFactory.php
$ git status
# ... a mesma coisa. Nada no staging.
```

Sem mensagens de erro. Sem warnings. O Git simplesmente fingia que o comando não existia. Nem `git commit -a` nem `git commit --include <arquivo>` surtiam efeito. O staging estava congelado de forma silenciosa.

O Diagnóstico da IA (E por que as soluções padrão falharam)

Hipótese 1: "Isso é regra do seu `.gitignore`"

A suspeita inicial de qualquer ferramenta. Bastou uma checagem rápida para descartá-la:

```
$ git check-ignore -v code/back/database/factories/DGTransactionFactory.php
# Sem output — o arquivo não estava sendo ignorado pelo Git.
```

Hipótese 2: "São os line endings (`.gitattributes`)"

O arquivo `code/back/.gitattributes` continha a regra `* text=auto eol=lf`. A IA formulou uma teoria complexa de que a normalização de quebras de linha (`LF` vs `CRLF`) estava impedindo o Git de reconhecer a alteração física. Bonito na teoria, mas inútil na prática.

Hipótese 3: "O Workaround com `update-index`"

Frustrada por não conseguir staging comum, a IA sugeriu um atalho de baixo nível:

```
$ git update-index --add code/back/database/factories/DGTransactionFactory.php
```

Funcionou. O arquivo foi para a área de staging e pôde ser commitado. Porém, isso remediou o sintoma, não a causa. Bastava dar um `git pull` ou trocar de branch para que o arquivo voltasse a ficar travado como "modificado" no `git status`.

Hipótese 4: `GIT_TRACE2` – A Investigação Profunda

Aqui a IA mostrou seu verdadeiro valor de análise técnica de baixo nível. Ela decidiu depurar os logs internos do Git em nível de chamada de sistema:

```
$ GIT_TRACE2_EVENT=/tmp/git-trace.json git add code/back/database/factories/DGTransactionFactory.ph
```

Ao analisar o JSON de eventos gerado pelo Git, a IA identificou a seguinte anomalia:

```
{"event":"data","key":"preload/sum_lstat","value":"1"}
{"event":"data","key":"preload/nr-lstat-saved","value":"1076"}
```

> ℹ️ **NOTE**
>
> **O que isso significa?**
> De um total de 1.077 arquivos no index, o Git fez apenas **1 chamada física** `lstat` ao disco. As outras 1.076 entradas foram resolvidas diretamente através do **stat cache**. O Git estava "convencido" de que sabia o estado exato dos arquivos e simplesmente ignorou as modificações reais no disco.

Investigando ainda mais a fundo o cache indexado, encontramos duplicatas fantasmas usando o comando `git ls-files`:

```
$ git ls-files | grep "factories/Dg[A-Z]"
# DgPaymentWebhookEventFactory.php
# DgPlanFactory.php
# DgProfileBoostFactory.php
# DgTransactionFactory.php
```

Havia duas entradas no index do Git para os mesmos arquivos físicos: uma com `DG...` (correta) e outra com `Dg...` (fantasma, com 'g' minúsculo), ambas apontando para hashes de objeto diferentes.

A Causa Raiz: macOS + Git + Case-Insensitive Filesystem

O macOS usa, por padrão, o sistema de arquivos APFS (ou HFS+ em sistemas antigos), que é **case-insensitive** (ignora maiúsculas e minúsculas ao buscar caminhos, tratando `DG...` e `Dg...` como o mesmo arquivo físico), mas **case-preserving** (mantém o case original na exibição).

Por outro lado, o Git é **case-sensitive** por padrão internamente. Em algum momento do histórico do projeto, houve uma alteração no case de uma pasta ou arquivo (uma transição de `Dg` para `DG`) que não foi adequadamente processada pelo Git no macOS.

Isso gerou uma **ghost entry** (entrada fantasma) no index do repositório: 1. O Git armazena metadados como `mtime` (data de modificação), `ctime` (data de criação) e `inode` para cada entrada do index. 2. Ao executar `git add`, ele compara os dados do disco com o cache (`stat cache`). 3. Como o sistema do macOS é case-insensitive, a leitura física do arquivo no disco coincidia exatamente com a entrada fantasma (`Dg...`). 4. Satisfeito com esse "hit" no cache de stat, o Git assumia que nada havia mudado e saltava a verificação do arquivo correto (`DG...`).

A IA desvendou toda essa intrincada engenharia interna. Mas ela empacou aí. Não sabia como sair do labirinto sem corromper o repositório ou sugerir reinstalações manuais de ferramentas.

A Solução: "Memória de Guerra" e o Round-Trip de Rename

Com o diagnóstico em mãos, recorri a um truque antigo — daqueles aprendidos nos primórdios da carreira, lidando com repositórios que transitavam entre servidores Linux (case-sensitive) e estações Windows/macOS (case-insensitive).

Disse à IA: *"Vamos forçar uma quebra de ambiguidade com um caractere neutro, commitar, voltar o nome e limpar o index."*

A receita é simples e infalível:

```
# Passo 1: Renomear para um nome temporário sem ambiguidades de case
$ git mv DGTransactionFactory.php D_GTransactionFactory.php
$ git commit -m "chore: renomeação temporária para limpar cache do Git"

# Passo 2: Renomear de volta para a nomenclatura correta
$ git mv D_GTransactionFactory.php DGTransactionFactory.php
$ git commit -m "chore: restaurando nomenclatura correta"

# Passo 3: Remover explicitamente a entrada fantasma (ghost entry) que restou no index
$ git rm --cached DgTransactionFactory.php
$ git commit -m "chore: removendo entrada fantasma do index"
```

Por que isso funciona?

O uso do caractere intermediário (`D_G...`) força o sistema de arquivos e o Git a concordarem que houve uma alteração real, quebrando o ciclo do stat cache e gerando uma entrada limpa e única. O comando `git rm --cached Dg...` garante que a entrada fantasma (com a grafia antiga) que estava travando o stat cache seja permanentemente deletada da base de dados do Git.

Após o round-trip de renames, o `git add` voltou a funcionar perfeitamente e os arquivos finalmente foram staged.

Por que este bug é tão traiçoeiro?

- **Silêncio total:** Sem avisos, sem mensagens de erro e um exit code `0` que mascara o problema para scripts de automação, hooks e IDEs.
- **Falsa sensação de integridade:** Novos arquivos criados no repositório funcionam perfeitamente com `git add`, fazendo parecer que o problema é local ou isolado de algum arquivo específico.
- **Invisível no fluxo de trabalho tradicional:** Comandos comuns como `git status`, `git diff` ou `git log` ocultam as duplicatas do index. Apenas uma análise refinada com `git ls-files` traz a verdade à tona.
- **Dependência de ambiente:** Em sistemas Linux puros (que são estritamente case-sensitive), esse bug simplesmente não ocorre, o que torna sua reprodução em pipelines de CI/CD impossível.

Conclusão e Dica de Prevenção

Se você desenvolve no macOS ou Windows e trabalha com nomenclaturas em CamelCase, vale a pena rodar este comando preventivo de vez em quando para rastrear duplicatas no index antes que elas causem problemas silenciosos:

```
$ git ls-files | sort | uniq -i -d
```

Se esse comando retornar alguma linha, você tem candidatas a ghost entries assombrando seu repositório.

> ℹ️ **NOTE**

Sobre o Autor

Maicon Santos (Webmaicon) é Tech Lead, AI Solutions Architect e desenvolvedor Full Stack com 27 anos de experiência em tecnologia. Pioneiro em segurança digital (tendo criado um sistema de 2FA via SMS em 2003) e no desenvolvimento de metodologias próprias de Prompt Engineering e Contextos de IA baseados em estruturas de skills e guidelines. Liderou equipes de tecnologia em grandes corporações como TV Globo, L'Oréal e Magazine Luiza, além de ser autor de blog técnico com mais de 320 mil acessos.

Caso Git Case-Sensitivity - AI Git Case Sensitivity MacOS